EGEE

EGEE gLite User's Guide

gLite I/O

Document identifier:	**EGEE-TECH-570771-v1.0**
Date:	**March 23, 2005**
Activity:	**JRA1: Data Management**
Document status:	**DRAFT**
Document link:	**https://edms.cern.ch/document/570771**

Abstract: This is the user's guide to the gLite Input/Output libraries.

Document Change Log

Issue	Date	Comment	Author
1.0	1.3.2005	Initial Version.	Text by Paolo Badino and Peter Kunszt

Document Change Record

Issue	Item	Reason for Change

Copyright ©Members of the EGEE Collaboration. 2004. See http://eu-egee.org/partners for details on the copyright holders.

EGEE ("Enabling Grids for E-science in Europe") is a project funded by the European Union. For more information on the project, its partners and contributors please see http://www.eu-egee.org.

You are permitted to copy and distribute verbatim copies of this document containing this copyright notice, but modifying this document is not allowed. You are permitted to copy this document in whole or in part into other documents if you attach the following reference to the copied elements: "Copyright ©2004. Members of the EGEE Collaboration. http://www.eu-egee.org"

The information contained in this document represents the views of EGEE as of the date they are published. EGEE does not guarantee that any information contained herein is error-free, or up to date.

EGEE MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, BY PUBLISHING THIS DOCUMENT.

CONTENTS

1. INTRODUCTION **5**
1.1. SERVICE ARCHITECTURE . . . 5
1.2. INTERACTIONS WITH OTHER SERVICES . . . 5

2. QUICKSTART GUIDE **5**
2.1. COMMAND LINE QUICKSTART . . . 5
2.2. API QUICKSTART: ACCESSING A REMOTE FILE USING GLITE I/O . . . 6

3. REFERENCE GUIDE **8**
3.1. CONFIGURATION . . . 8
3.1.1. CONFIGURATION PARAMETERS . . . 8
3.1.2. CONFIGURATION FILE . . . 9
3.2. COMMANDLINE INTERFACES . . . 9
3.2.1. GLITE-PUT: UPLOAD A LOCAL FILE TO THE STORAGE ELEMENT. . . . 10
3.2.2. GLITE-GET: COPY A REMOTE FILE FROM THE STORAGE ELEMENT TO A LOCAL FILE. . . . 10
3.2.3. GLITE-RM: REMOVE A REMOTE FILE FROM THE STORAGE ELEMENT. 11
3.2.4. CONFIGURATION . . . 11
3.3. APPLICATION PROGRAM INTERFACES . . . 11
3.3.1. GLITE_OPEN . . . 11
3.3.2. GLITE_CREAT . . . 12
3.3.3. GLITE_READ . . . 13
3.3.4. GLITE_WRITE . . . 13
3.3.5. GLITE_LSEEK . . . 13
3.3.6. GLITE_FSTAT . . . 14
3.3.7. GLITE_CLOSE . . . 14
3.3.8. GLITE_UNLINK . . . 15
3.3.9. GLITE_ERROR . . . 15
3.3.10. GLITE_STRERROR . . . 15
3.3.11. GLITE_POSIX_OPEN . . . 16
3.3.12. GLITE_POSIX_CREAT . . . 16
3.3.13. GLITE_POSIX_READ . . . 17
3.3.14. GLITE_POSIX_WRITE . . . 17
3.3.15. GLITE_POSIX_LSEEK . . . 17
3.3.16. GLITE_POSIX_LSEEK64 . . . 18

3.3.17. GLITE_POSIX_FSTAT 18
3.3.18. GLITE_POSIX_FSTAT64 19
3.3.19. GLITE_POSIX_CLOSE 19
3.3.20. GLITE_POSIX_UNLINK 19
3.3.21. GLITE_FILEHANDLE 20
3.3.22. GLITE_POSIX_ERRNO 20
3.3.23. GLITE_STAT 21
3.3.24. TYPE DEFINITIONS AND CONSTANTS 22

4. KNOWN PROBLEMS AND CAVEATS 22
4.1. CASTOR TIMEOUT 22

5. COMPONENT AND INTERACTION DIAGRAMS 22

1. INTRODUCTION

Input/Output librares are used to access files for read and write. This is the most basic operation in any system dealing with data stored in files. The files on the Grid are known to the user by its Logical File Name (LFN). The gLite I/O library provides the necessary methods and command-line tools to interact with the grid storage using LFNs. To get a complete overview of the gLite data management services and how the I/O fits into them, as well as explanations about file names etc, the reader is referred to the Overview of gLite Data Management User's Guide [2].

1.1. SERVICE ARCHITECTURE

From the user's point of view, gLite I/O is just a set of client libraries that can be used like a posix I/O library. The server-side architecture determines the semantics of the library to a large extent.

The sequence diagrams in the appendix 5. show how the client interacts with the server for the four most basic operations.

1.2. INTERACTIONS WITH OTHER SERVICES

The I/O client only interacts with the I/O server. In order to do so, the client needs a valid user VOMS proxy certificate, so there is an indirect dependency on VOMS or at least grid-proxy-init.

The I/O server interacts with the gLite File and Replica Catalog, the File Authorization Service, the SRM and the underlying storage to actually resolve the file name and to serve up the bytes to the user.

2. QUICKSTART GUIDE

2.1. COMMAND LINE QUICKSTART

The gLite I/O library comes with a set of command line tools to facilitate the user interaction with the grid. In order to put a local file 'into the grid' the `glite-put` command can be used:

```
UI> glite-put mylocalfile lfn:///tmp/myfile1
[glite_put] Total 0.00 MB       |====================| 100.00 % [0.0 Mb/s]

Transfer Completed:

  LFN                    : /tmp/myfile1
  GUID                   : 008bf98e-27aa-1227-be5e-898a04a1beef
  SURL                   : srm://gridftp05.cern.ch:8443/srm/managerV1?SFN=/castor/cern.ch/use
  Data Written [bytes]   : 375
  Eff.Transfer Rate[Mb/s] : 0.000094
```

The three slash notation is in accordance with RFC 2396 [1]. Instead, the user can also specify just `/tmp/myfile1`. The verbosity of glite-put can be suppressed by using the `-s` *silence* option. In the same way, data files can be retrieved from the Grid by issuing `glite-get`:

```
UI> glite-get lfn:///tmp/myfile1 myfile.grid glite_get] Total 0.00 MB
|====================| 100.00 % [0.0 Mb/s]
```

```
Transfer Completed:

  LFN                     : /tmp/myfile1
  GUID                    : 008bf98e-27aa-1227-be5e-898a04a1beef
  SURL                    : srm://gridftp05.cern.ch:8443/srm/managerV1?SFN=/castor/cern.ch/use
  Data Written [bytes]    : 375
  Eff.Transfer Rate[Mb/s] : 0.000041
```

Finally, the data can be removed by issuing `glite-rm`

```
UI> glite-rm lfn:///tmp/myfile1
```

The server that is being contacted is specified in the configuration, see Section 3.1. For more details on these command-line tools please see the reference guide below in Section 3.2.

2.2. API QUICKSTART: ACCESSING A REMOTE FILE USING GLITE I/O

In order to access a remote file, you need to write your own application linking the gLite I/O client library (`libglite_data_io_client.so`) and opening the file the same way you open a local file using POSIX calls. You may use either the `glite_posix_*` APIs, in which case the syntax of open, close, read, write and lseek operations is exaclty the same of the standard POSIX methods, or you may choose to use the `glite_*` API calls, which differ from the POSIX ones since they use gLite defined types and accept additional parameters mainly related to optimization and thread-safe error reporting. The flow will always be the same in both cases.

Reading a file using extended interface:

```
#include "glite/data/io/client/ioclient.h"

glite_result gl_res;
glite_error  err;
glite_handle fh;

fd = glite_open(remotefilename,O_RDONLY,0,0,&gl_res);
// error handling
if(glite_handle == GLITE_NULL_HANDLE) {
    err = glite_error(fh);
    fprintf(stdout,"Cannot open file: %s\n",glite_strerror(err));
    // return or exit or ...
}

int nread = 0;
char buffer[TRANSFERBLOCKSIZE];
do {
    nread =  glite_read(fh,buffer,TRANSFERBLOCKSIZE);
    // error handling
    if(nread < 0) {
       err = glite_error(fh);
       fprintf(stdout,"Cannot read file: %s\n",glite_strerror(err));
```

```
        // break or return or exit or ...
    }
    //  Do something ...
} while (nread > 0);
glite_close(fh);
```

Reading a file using POSIX interface:

```
#include <string.h>
#include "glite/data/io/client/ioclient-posix.h"

int nread = 0;
char buffer[TRANSFERBLOCKSIZE];
int fd;

fd = glite_posix_open(remotefilename,O_RDONLY,0);
// error handling
if(fd < 0) {
    fprintf(stdout,"Cannot open file: %s\n",strerror(glite_posix_errno));
    // return or exit or ...
}

do {
    nread =  glite_posix_read(fd,buffer,TRANSFERBLOCKSIZE);
    // error handling
    if(nread < 0) {
       fprintf(stdout,"Cannot read file: %s\n",strerror(glite_posix_errno));
       // break or return or exit or ...
    }
    //  Do something ...
} while (nread > 0);
glite_posix_close(fd);
```

The remotefilename is a logical file name like in the command line example above, e.g. `lfn:///tmp/myfile1`. The flag is a bitwise or from the system values `O_RDONLY`, `O_WRONLY`, `O_CREAT`, while mode specifies the permissions the file should have upon creation. Upon success, a file descriptor is returned, otherwise the result is less than zero and errors have to be handled. In the posix case there is a `glite_posix_errno` variable defined which may be used just like the usual posix `errno` variable. Once the file has been opened say for read, it can be accessed by the posix lseek and read methods to get blocks of data directly. For more details on each method, see the reference guide below.

NOTE: Beware that the usage of the `glite_posix_errno` variable is not thread-safe. Don't use the posix interface in threaded programs.

In order to compile and link an application that uses glite-io, the following flags have to be passed in to the compiler and linker respectively:

```
Compilation:

-I$GLITE_LOCATION/include
```

```
Linkage:

GLOBUS_FLAVOR_THREADS=gcc32dbg
LIBS=-L$(GLITE_LOCATION)/lib -lglite_data_io_client \
-L$(GLOBUS_LOCATION)/lib \
-lglobus_gsi_proxy_core_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_gsi_credential_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_gsi_callback_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_oldgaa_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_gsi_sysconfig_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_gsi_cert_utils_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_openssl_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_proxy_ssl_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_openssl_error_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_ftp_client_$(GLOBUS_FLAVOR_THREADS) \
-lssl_$(GLOBUS_FLAVOR_THREADS) \
-lcrypto_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_gss_assist_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_gssapi_gsi_$(GLOBUS_FLAVOR_THREADS) \
-lglobus_common_$(GLOBUS_FLAVOR_THREADS)
```

3. REFERENCE GUIDE

The package providing the gLite I/O client is called `org.glite.data.io-client`. It contains the command line tools as well as the API libraries and include files.

3.1. CONFIGURATION

3.1.1. CONFIGURATION PARAMETERS

Unless a different configuration namespace is used (see the INSTALL file coming with the org.glite.data.io-client package, "Creating Different Configurations"), the gLite I/O client initializes itself at the first `glite_open` (or `glite_posix_open`) call, looking at the configuration files in named `glite-io-client.properties.xml` and `glite-io-client.log-properties` in `$GLITE_LOCATION_USER/etc`, `$GLITE_LOCATION/etc` or `$GLITE_LOCATION_VAR/etc`. That configuration file should contain at least the entries for the glite-io-client component.

Configuration parameters used at initialization time are:

Server **Mandatory** The name of the host where the gLite I/O server is running.

ServerPort The port number that the gLite I/O server is listening on. Default value: 9999.

EncryptName Enable encryption of the file name. Allowed values are "true" and "false". Default value: true.

EncryptData Enable encryption of the data blocks sent and received. Allowed values are "true" and "false". Default value: false.

NumberOfStreams Number of QUANTA tcp streams. Default value: 1.

CacheLevel AliEn aiod Cache Level value. Default value: 7.

3.1.2. CONFIGURATION FILE

This section contains an example of the gLite I/O client configuration file. All the hostname and pathname values should be changed according to the environment where the gLite IO client is installed.

Content of `glite-io-client.properties.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<service name="glite-io-client">
  <components>
    <component name="io-client">
      <init>
        <param><name>Server</name>      <value>__hostname__</value></param>
        <param><name>ServerPort</name>  <value>9999</value></param>
        <param><name>EncryptName</name> <value>true</value></param>
        <param><name>EncryptData</name> <value>true</value></param>
      </init>
    </component>
  </components>
</service>
```

An example of a logging configuration file `glite-io-client.log-properties` is:

```
log4j.rootCategory=DEBUG, file

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%m%n

log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.fileName={HOME}/.glite/var/log/glite-io-client.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d %-6p %c - %m%n
```

3.2. COMMANDLINE INTERFACES

There are three CLI tools that can be used to upload/download/remove files into/from a Storage Element. These three tools are:

`glite-get` Retrieve a file from the local Storage Element.

`glite-put` Upload a file to the Storage Element.

`glite-rm` Remove a file.

They all have the following options in common:

`-h` Print a help screen.

`-s` Be silent.

`-c <name>` Use configuration with name `<name>`, overriding the default glite-io-client configuration.

By default, the `glite-get` and `glite-put` command line tools will display the standard error a progress bar showing the status of the transfer. For all three tools a report is printed on completion, showing information about the transferred (or removed) file. Using the `-s` option instructs the command line tools to suppress these informational messages. Errors are still printed to stderr.

3.2.1. GLITE-PUT: UPLOAD A LOCAL FILE TO THE STORAGE ELEMENT.

The syntax for the put command is:

```
      glite-put <localfilename> lfn://<lfn>
or
      glite-put <localfilename> <lfn>
```

where `<localfilename>` is the name of the local file you want to upload and `<lfn>` is the logical file name you want to assign to that file. Please remember that a valid LFN should always start with a '/', which in the first (URL) form results in three consecutive slashes [1].

In case you want to assign a GUID, you can use the syntax

```
      glite-put <localfilename> guid:<guid>
or
      glite-put <localfilename> <guid>
```

where `<guid>` is the GUID you want to assign to the file. The GUID has always to be a valid GUID string representation (36 bytes that must follow the format string representation `"%08x-%04x-%04x-%04x-%012x"`)

If you want to specify explicitly the permission bits, you can use the `-m` option

```
      glite-put <localfilename> lfn://<lfn> -m <mode>
```

where `<mode>` should be an octal integer in the range `[0001,0777]`. The default value is `0640`.

3.2.2. GLITE-GET: COPY A REMOTE FILE FROM THE STORAGE ELEMENT TO A LOCAL FILE.

The syntax is:

```
      glite-get lfn://<lfn> <localfilename>
or
      glite-get <lfn> <localfilename>
```

where `<lfn>` is the logical file name of the file you want to download and `<localfilename>` is the name of the destination local file

In case you want to use GUIDs, you can use the syntax

```
      glite-get guid:<guid> <localfilename>
or
      glite-get <guid> <localfilename>
```

where `<guid>` is the GUID of the remote file

3.2.3. GLITE-RM: REMOVE A REMOTE FILE FROM THE STORAGE ELEMENT.

If the given file is the last replica then also the Logical File Name and the GUID will be deleted from the catalog.

```
glite-rm lfn://<lfn>
or
glite-rm <lfn>
```

where `<lfn>` is the logical file name of the file you want to remove In case you want to use GUIDs, you can use the syntax

```
glite-rm guid:<guid>
or
glite-rm <guid>
```

where `<guid>` is the GUID of the remote file.

By default, that application displays to the standard error some information concerning the remove action. If you want to run that application in silent mode, you have to use the option `-s`: in that case only errors will be printed to stderr.

3.2.4. CONFIGURATION

In order to use the command line tools you need to set up the environment and configure glite-io-client. To set the environment up, you need to acquire your Grid Proxy Certificate and then set the $LD_LIBRARY_PATH environment variable in order to locate the glite io client library and its dependencies. Normally this is done automatically on a Grid User Interface and Worker Node. Details on how to do this can be found in the gLite Installation Guide.

In case you want to override the default glite-io-cient configuration values, you have to create a pair of configuration files (for example `glite-io-client-customconfig.properties.xml` and `glite-io-client-customconfig.log-properties`) and store them in one of the predefined configuration paths (`$GLITE_LOCATION_USER/etc`, `$GLITE_LOCATION/etc` or `$GLITE_LOCATION_VAR/etc`), filling them with the Glite IO Client configuration parameters you want to assign and then invoke the transfer applications by using the `"-c <config-name>"` option. In our example, you'll have to type:

```
glite-put -c glite-io-client-customconfig <localfilename> lfn://<lfn>

glite-get -c glite-io-client-customconfig lfn://<lfn> <localfilename>

glite-rm  -c glite-io-client-customconfig lfn://<lfn>
```

3.3. APPLICATION PROGRAM INTERFACES

3.3.1. GLITE_OPEN

Open a Remote File. The parameters are:

`pathname` *IN* the name of the remote file to open. It can be in the format `lfn://<file_name>` or `guid:<file_guid>`. `glite_open` also accepts the file name without the prefix: In that case if

```
#include "glite/data/io/client/ioclient.h"

glite_handle glite_open(const char *  pathname,
                        glite_int32   flags,
                        glite_int32   mode,
                        glite_int64   size,
                        glite_result* result);
```

`file_name` starts with a '/' is interpreted as an LFN, otherwise it will assumed it is a GUID and a GUID representation validity check will be performed on it.

`flags` *IN* the requested access flag. The value should be a the bitwise inclusive-OR of values from the following lists:

`O_RDONLY` open for reading only

`O_WRONLY` open for writing only

`O_CREAT` create a file

`mode` *IN* specifies what permissions the file has when it is created

`size` *IN* specifies how many bytes should be preallocated remotely. It will be used only if the `O_CREAT` flag is set and may be null

`result` *OUT* the result code associated with the operation. This parameter is used to return additional information in case of errors and can be null if that information is not requested.

Return value: a handle to the opened file, or `GLITE_NULL_HANDLE` in case of failure.

3.3.2. GLITE_CREAT

```
#include "glite/data/io/client/ioclient.h"

glite_handle glite_creat(const char *  pathname,
                         glite_int32   mode,
                         glite_int64   size,
                         glite_result* result);
```

Create a Remote File. This call is equivalent to

```
glite_open(name, O_CREAT, mode, size, result)
```

The parameters are:

`pathname` *IN* the name of the remote file to open. It can be in the format `lfn://<file_name>` or `guid:<file_guid>`. `glite_creat` also accepts the file name without the prefix: In that case if `file_name` starts with a '/' is interpreted as an LFN; otherwise if it's a valid GUID string representation, it's interpreted as a GUID.

`mode` *IN* specifies what permissions the file has when it is created.

`size` *IN* specifies how many bytes should be preallocated remotely. May be null.

`result` *OUT* the result code associated with the operation. This parameter is used to return additional information in case of errors and can be null if that information is not requested.

Return value: a handle to the opened file, or `GLITE_NULL_HANDLE` in case of failure.

3.3.3. GLITE_READ

```
#include "glite/data/io/client/ioclient.h"

glite_int32 glite_read(glite_handle fh, void * buf, glite_int32 count);
```

Read the file.The parameters are:

`fh` *IN* the file handle returned by a previos `glite_open` call

`buf` *IN* the buffer where to store the data

`count` *IN* the number of byte to read. This parameter should be equal or less the size of the buffer.

Return value: the number of bytes read, or a negative value in case of error

3.3.4. GLITE_WRITE

```
#include "glite/data/io/client/ioclient.h"

glite_int32 glite_write(glite_handle fh, const void * buf, glite_int32 count);
```

Write to the file. The parameters are:

`fh` *IN* the file handle returned by a previos `glite_open` or `glite_creat` call

`buf` *IN* the buffer containing the data to be written

`count` *IN* the number of byte to write. This parameter should be equal or less the size of the buffer

Return value: the number of bytes written, or a negative value in case of error

3.3.5. GLITE_LSEEK

Set a file read/write pointer. The parameters are:

`fh` *IN* the file handle returned by a previos `glite_open` or `glite_creat` call

```
#include "glite/data/io/client/ioclient.h"

glite_int64 glite_lseek(glite_handle fh, glite_int64 offset, glite_int32 whence);
```

`offset` *IN* new byte offset to seek to

`whence` *IN* relative file position. it can be one of the following values:

- `SEEK_SET` set to offset
- `SEEK_CUR` set to current position plus offset
- `SEEK_END` set to the size of the file plus offset

Return value: The new offset from the beginning of the file, or a negative value in case of error

3.3.6. GLITE_FSTAT

```
#include "glite/data/io/client/ioclient.h"

glite_int32 glite_fstat(glite_handle fh, struct glite_stat * buf);
```

Get Information about the file associated with the given handle. The parameters are:

`fh` *IN* the file handle returned by a previos `glite_open` or `glite_creat` call

`buf` *OUT* pointer to a stat structure, as defined in iotypes.h, into which information is placed concerning the file

Return value: `GLITE_IO_SUCCESS` if success, or a negative value in case of error

3.3.7. GLITE_CLOSE

```
#include "glite/data/io/client/ioclient.h"

glite_int32 glite_close(glite_handle fh);
```

Close the file. The parameters are:

`fh` *IN* the file handle returned by a previos `glite_open` or `glite_creat` call

Return value: `GLITE_IO_SUCCESS` if success, a negative value in case of error

```
#include "glite/data/io/client/ioclient.h"

glite_int32 glite_unlink(const char * pathname);
```

3.3.8. GLITE_UNLINK

Deletes a file on the remote Storage element. That replica information will be removed from the Logical File System and an attempt is made to remove the physical instance, but if that fails no error are returned. In case that replica is the last one, also the file logical name and GUID will be removed, so they can be reused later The parameters are:

`pathname` *IN* the name of the remote file to delete. It could be in the format `lfn://<file_name>` or `guid:<file_guid>`. `glite_unlink` also accept the file name without the prefix: In that case if file_name starts with a '/' is interpreted as an LFN; otherwise if it's a valid GUID string representation, it's interpreted as a GUID

Return value: `GLITE_IO_SUCCESS` if success, a negative value in case of error

3.3.9. GLITE_ERROR

```
#include "glite/data/io/client/ioclient.h"

glite_int32 glite_error(glite_handle fh);
```

Get the Last GLite Error code associated to a File Handle. **NOTE:** This method cannot be used after an open or close call, since there's no valid file handle. The parameters are:

`fh` *IN* the file handle returned by a previos `glite_open` or `glite_creat` call

Return value: `GLITE_IO_SUCCESS` if success, a negative value in case of error

3.3.10. GLITE_STRERROR

```
#include "glite/data/io/client/ioclient.h"

const char * glite_strerror(glite_result error);
```

Get a string representing a GLite Error code. The parameters are:

`error` *IN* the error code to be translated

Return value: the error strig

```
#include "glite/data/io/client/ioclient-posix.h"

int glite_posix_open(const char * pathname, int flags, mode_t mode);
```

3.3.11. GLITE_POSIX_OPEN

Open a Remote File. The parameters are:

`pathname` *IN* the name of the remote file to open. It can be in the format `lfn://<file_name>` or `guid:<file_guid>`. `glite_posix_open` also accepts the file name without the prefix: In that case if file_name starts with a '/' is interpreted as an LFN, otherwise it will assumed it is a GUID and a GUID representation validity check will be performed on it.

`flags` *IN* the requested access flag. The value should be a the bitwise inclusive-OR of values from the following lists:

- `O_RDONLY` open for reading only
- `O_WRONLY` open for writing only
- `O_CREAT` create a file

`mode` *IN* specifies what permissions the file has when it is created.

Return value: On success, the new file descriptor. On error -1 is returned and `glite_posix_errno` is set appropriately.

3.3.12. GLITE_POSIX_CREAT

```
#include "glite/data/io/client/ioclient-posix.h"

int glite_posix_creat(const char * pathname, mode_t mode);
```

Create a Remote File. This call is equivalent to

```
glite_posix_open(pathname, O_CREAT, mode)
```

The parameters are:

`pathname` *IN* the name of the remote file to open. It could be in the format `lfn://<file_name>` or `guid:<file_guid>`. `glite_creat` also accept the file name without the prefix: In that case if file_name starts with a '/' is interpreted as an LFN; otherwise if it's a valid GUID string representation, it's interpreted as a GUID

`mode` *IN* specifies what permissions the file has when it is created

Return value: On success, the new file descriptor. On error, -1 is returned, and `glite_posix_errno` is set appropriately

```
#include "glite/data/io/client/ioclient-posix.h"

ssize_t glite_posix_read(int fd, void * buf, size_t count);
```

3.3.13. GLITE_POSIX_READ

Read up to count bytes from file descriptor fd into the buffer starting at buf. The parameters are:

`fd` *IN* the file descriptor returned by a previos `glite_posix_open` call

`buf` *IN* the buffer where to store the data

`count` *IN* the number of byte to read. This parameter should be equal or less the size of the buffer

Return value: On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. On error, -1 is returned, and `glite_posix_errno` is set appropriately

3.3.14. GLITE_POSIX_WRITE

```
#include "glite/data/io/client/ioclient-posix.h"

ssize_t glite_posix_write(int fd, const void * buf, size_t count);
```

Writes up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf. The parameters are:

`fd` *IN* the file descriptor returned by a previos `glite_posix_open` or `glite_posix_creat` call

`buf` *IN* the buffer containing the data to be written

`count` *IN* the number of byte to write. This parameter should be equal or less the size of the buffer

Return value: On success, the number of bytes written are returned (zero indicates nothing was written). On error, -1 is returned, and `glite_posix_errno` is set appropriately.

3.3.15. GLITE_POSIX_LSEEK

```
#include "glite/data/io/client/ioclient-posix.h"

off_t glite_posix_lseek(int fd, off_t offset, int whence);
```

Set a file read/write pointer. The parameters are:

`fd` *IN* the file descriptor returned by a previos `glite_posix_open` or `glite_posix_creat` call

`offset` *IN* new byte offset to seek to

`whence` *IN* relative file position. it can be one of the following values:

- `SEEK_SET` set to offset
- `SEEK_CUR` set to current position plus offset
- `SEEK_END` set to the size of the file plus offset

Return value: the resulting offset location as measured in bytes from the beginning of the file. Otherwise, a value of `(off_t)-1` is returned and `glite_posix_errno` is set to indicate the error

3.3.16. GLITE_POSIX_LSEEK64

```
#include "glite/data/io/client/ioclient-posix.h"

off64_t glite_posix_lseek64(int fd, off64_t offset, int whence);
```

Set a file read/write pointer. The parameters are:

`fd` *IN* the file descriptor returned by a previos `glite_posix_open` or `glite_posix_creat` call

`offset` *IN* new byte offset to seek to

`whence` *IN* relative file position. it can be one of the following values:

- `SEEK_SET` set to offset
- `SEEK_CUR` set to current position plus offset
- `SEEK_END` set to the size of the file plus offset

Return value: the resulting offset location as measured in bytes from the beginning of the file. Otherwise, a value of `(off64_t)-1` is returned and `glite_posix_errno` is set to indicate the error

3.3.17. GLITE_POSIX_FSTAT

```
#include "glite/data/io/client/ioclient-posix.h"

int glite_posix_fstat(int fd, struct stat *buf);
```

Return information about the specified file. The parameters are:

`fd` *IN* the file descriptor returned by a previos `glite_posix_open` or `glite_posix_creat` call

`buf` *OUT* pointer to a stat structure, as defined in <sys/stat.h>, into which information is placed concerning the file

Return value: On success, zero is returned. On error, -1 is returned, and `glite_posix_errno` is set appropriately

```
#include "glite/data/io/client/ioclient-posix.h"

int glite_posix_fstat64(int fd, struct stat64 *buf);
```

3.3.18. GLITE_POSIX_FSTAT64

Return information about the specified file. To be used for large files. The parameters are:

`fd` *IN* the file descriptor returned by a previos `glite_posix_open` or `glite_posix_creat` call

`buf` *OUT* pointer to a stat structure, as defined in <sys/stat.h>, into which information is placed concerning the file

Return value: On success, zero is returned. On error, -1 is returned, and `glite_posix_errno` is set appropriately

3.3.19. GLITE_POSIX_CLOSE

```
#include "glite/data/io/client/ioclient-posix.h"

int glite_posix_close(int fd);
```

Close the file. The parameters are:

`fd` *IN* the file descriptor returned by a previos `glite_posix_open` or `glite_posix_creat` call

Return value: On success, zero is returned. On error, -1 is returned, and `glite_posix_errno` is set appropriately

3.3.20. GLITE_POSIX_UNLINK

```
#include "glite/data/io/client/ioclient-posix.h"

int glite_posix_unlink(const char * pathname);
```

Deletes a file on the remote Storage element. That replica information will be removed from the Logical File System and an attempt is made to remove the physical instance, but if that fails no error are returned. In case that replica is the last one, also the file logical name and GUID will be removed, so they can be reused later. The parameters are:

`pathname` *IN* the name of the remote file to delete. It could be in the format `lfn://<file_name>` or `guid:<file_guid>`. `glite_creat` also accept the file name without the prefix: In that case if file_name starts with a '/' is interpreted as an LFN; otherwise if it's a valid GUID string representation, it's interpreted as a GUID

Return value: On success, zero is returned. On error, -1 is returned, and `glite_posix_errno` is set appropriately

3.3.21. GLITE_FILEHANDLE

```
#include "glite/data/io/client/ioclient-posix.h"

glite_handle glite_filehandle(int fd);
```

Returns the file handle associated with the descriptor. The returned handle could be used to call glite-io non-posix methods (like `glite_fstat`), but calling `glite_close` on a file opened with `glite_posix_open` may cause an impredictable behavior, so should be avoided in any case. The parameters are:

`fd` *IN* the file descriptor returned by a previos `glite_posix_open` or `glite_posix_creat` call

Return value: On success, the associated file handle if success. On error, `GLITE_NULL_HANDLE` is returned

3.3.22. GLITE_POSIX_ERRNO

```
#include "glite/data/io/client/ioclient-posix.h"

extern int glite_posix_errno;
```

Number of last error. **Be aware that is not thread safe !**

3.3.23. GLITE_STAT

```
#include "glite/data/io/client/ioclient.h"

struct glite_stat {
    glite_int32  dev;        // The device
    glite_int64  ino;        // The inode on the remote
                             // Storage System
    glite_int32  mode;       // The file protection and type bits.
    glite_int32  nlink;      // The number of hard links on
                             // the remote Storage System
    glite_int32  uid;        // The user ID of owner on the
                             // remote Storage System
    glite_int32  gid;        // The group ID of owner on the
                             // remote Storage System
    glite_int32  rdev;       // The device type (if inode device)
    glite_int64  size;       // The total size, in bytes
    glite_int32  blksize;    // The blocksize for remote
                             // Storage Element filesystem I/O
    glite_int64  blocks;     // The number of blocks allocated
                             // on the remote Storage System
    glite_time_t atime;      // Time of last access
    glite_time_t mtime;      // Time of last modification
    glite_time_t ctime;      // Time of last change
    char guid[GLITE_GUID_LENGTH];  // File GUID
    char lfn[GLITE_LFN_LENGTH];    // The logical file name
    char surl[GLITE_SURL_LENGTH];  // The physical file name
    char owner[GLITE_NAME_LENGTH]; // The name of the owner
    char group[GLITE_NAME_LENGTH]; // The group name of owner
};
```

IO Stat Type. This structure is used to return File information using the `glite_fstat` method. The meaning of each field is the same as the posix `stat64` structure, which extends addition the guid, lfn, surl, owner and group fields.

3.3.24. TYPE DEFINITIONS AND CONSTANTS

```
typedef int       glite_int32;
typedef long long glite_int64;
typedef void *    glite_handle;
typedef int       glite_result;
typedef time_t    glite_time_t;

#define GLITE_NULL_HANDLE   0

#define GLITE_LFN_LENGTH    1024
#define GLITE_SURL_LENGTH   2048
#define GLITE_GUID_LENGTH   37
#define GLITE_NAME_LENGTH   1024
```

4. KNOWN PROBLEMS AND CAVEATS

4.1. CASTOR TIMEOUT

There is an as yet unresolved open issue concerning timeouts – an internal dependency of gLite-I/O has a hardcoded timeout of 5 minutes. If the SRM is a mass storage system like Castor where staging of data off tape can take longer than that, the connection will time out prematurely. Currently the only thing that can be done is that the client re-tries later. This will be resolved in future releases of gLite I/O.

5. COMPONENT AND INTERACTION DIAGRAMS

REFERENCES

[1] URI Syntactic Components. http://www.faqs.org/rfcs/rfc2396.html.

[2] JRA1 Data Management Cluster. *Overview of gLite Data Management*. EGEE, March 2005. https://edms.cern.ch/document/570643.

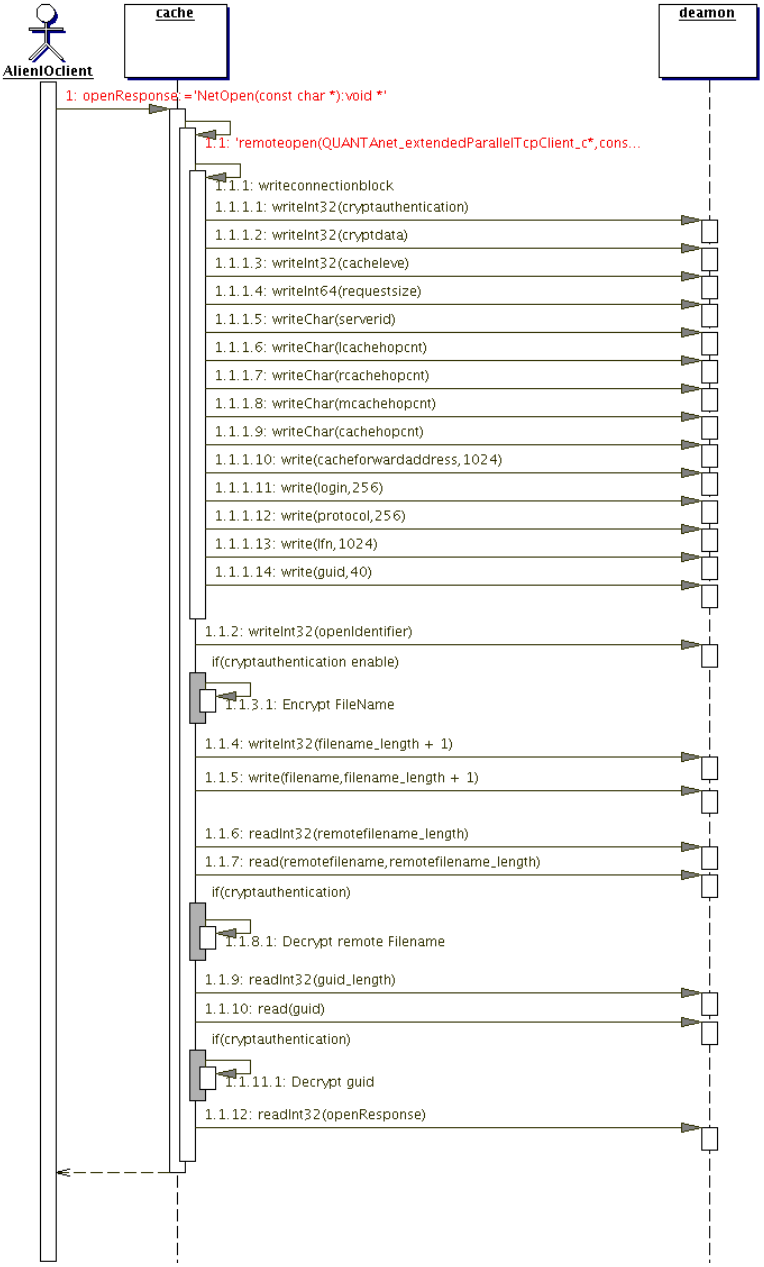

Figure 1: Sequence diagram showing the steps in a client-side gLite-I/O open call.

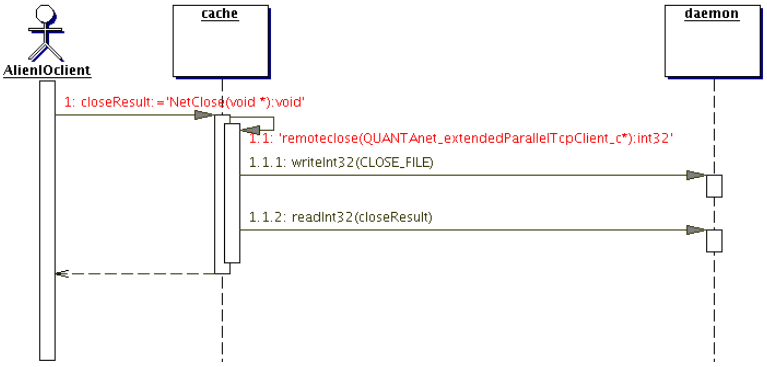

Figure 2: Sequence diagram showing the steps in a client-side gLite-I/O close call.

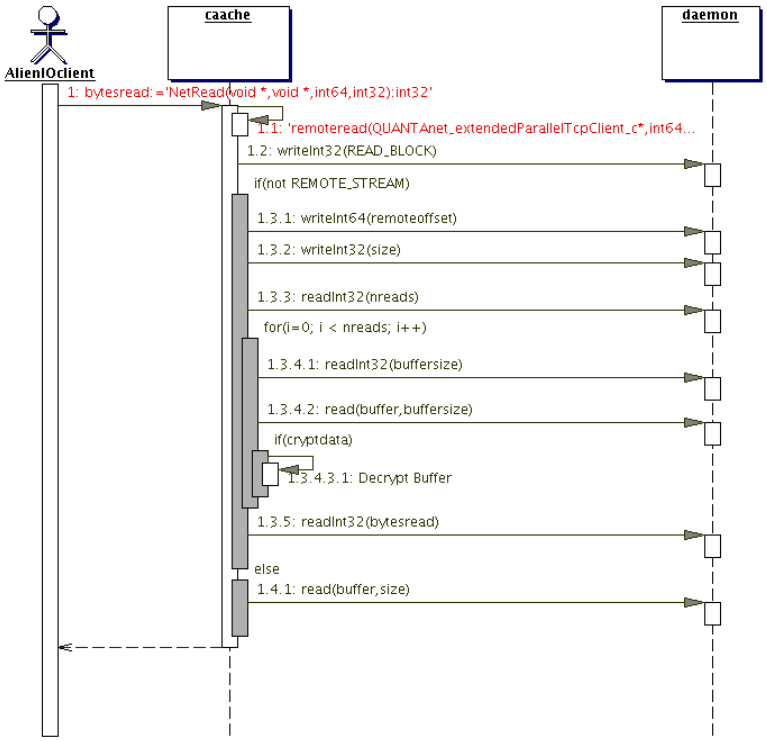

Figure 3: Sequence diagram showing the steps in a client-side gLite-I/O read call.

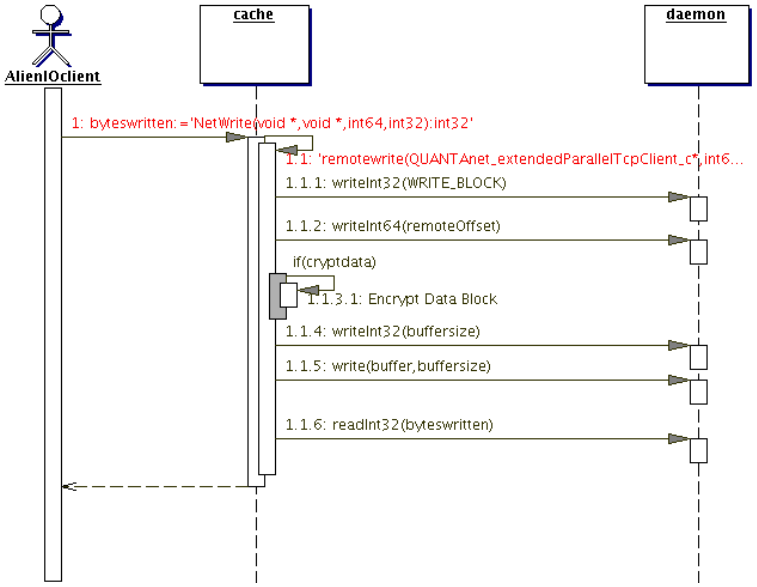

Figure 4: Sequence diagram showing the steps in a client-side gLite-I/O write call.